WGUPS Annotations

### Strengths (I1)

1. The algorithm orders package delivery by deadline, then the earliest load time, and then the nearest distance. This results in an algorithm that prioritizes deadline package delivery in the quickest possible time while maintaining efficient route creation.
2. The algorithm is simple and easy to visualize.

### Criteria (I2)

Delivers All Packages - ✓

The algorithm delivers all packages according to special notes in an acceptable time.

Deadline Packages - ✓

Algorithm delivers all deadline packages on time through 2 methods:

1. Prioritizes packages with a sooner deadline above all other packages.
2. Prioritizes packages with same deadline by nearest neighbor

Delayed Packages - ✓

Algorithm delivers delayed packages (w/ and w/out deadlines) by adding the package to the route based on the current route’s clock and what the earliest load time of the package is.

### Other algorithms (I3)

The following algorithms are heuristics used to solve the Traveling Salesman Problem (TSP) in a computationally efficient time compared to the brute force algorithm to find the optimal solution in O(N!) time.

1. Multi-Fragment Heuristic - Join all cities into route fragments where each fragment consists of 3 cities closest together. Continue to join all fragments into new fragments that do not result in a closed loop until all cities are connected to 2 other cities. My algorithm instead visits each node based on its nearest neighbor. I imagine the multi-fragment algorithm would be more difficult to implement with the complex package requirements given.

Multi-Fragment paraphrased from: <https://users.cs.cf.ac.uk/C.L.Mumford/howard/Introduction.html>

1. Farthest Insertion Heuristic - The farthest insertion algorithm operates by initially selecting the two points which are farthest from each other to form a route. Next, points are added based on their farthest distance from any point on the route. This continues until all points are part of the route. I think this would be difficult to determine which point is considered the “farthest” given the fact that packages have delays, deadlines, groupings, and truck assignments. I believe it is easier to take these into account with the Nearest Neighbor algorithm, which I use, visiting the nearest neighbor in accordance with package specifications.

Farthest Insertion paraphrased from: <https://users.cs.cf.ac.uk/C.L.Mumford/howard/Introduction.html>

### What would I do differently? (J)

One thing I might choose to change if I could re-do this project would be to create a heap data structure for the route building method (sortPackages()). This heap structure would allow me to call heapify() which would rearrange the stored packages to place the most eligible package on top of the structure in O(logN) time. This would allow me to build routes in O(NlogN) time with my nearest neighbor algorithm.

### My data structure (K1)

I used a hash table to store packages and a graph to represent the delivery locations.

The hash table operates at O(1) average case and O(N) worst case complexities. It devolves to O(N) anytime the data becomes not random and begins piling up in a single hash cell. Otherwise, the system will scale at O(1) with large amounts of randomized or sequential data.

The graph is created with a list of nodes and an adjacency matrix to represent edges. It is highly configurable and adjustable, and is especially useful for finding paths between nodes. It allows for easy “hot-swapping” of algorithms. The graph retrieves edges and nodes with a complexity of O(1). It adds edges with a complexity of O(1), and new nodes with a complexity of O(N). The primary cost of this data structure comes from the initial creation (O(N^2)).

### Other data structures (K2)

1. Binary Search Tree - This data structure creates a tree in which each node has 0-2 child nodes. Searching for an item takes O(LogN) time. Over time, the tree can grow imbalanced, decreasing overall search and insert efficiency, in which case it will need to be re-balanced. This is different from my hash table primarily in the efficiency of operation and in how you access the searched item. In the BST you search by visiting each node’s child (chosen by key being greater or less than current node) until you’ve found the item. The hash table generally takes 1 operation to find the item.
2. List (C-Array) - This data structure is built into C and python and is inefficient because it takes O(N) time to search and insert. You have to traverse the whole data structure just to find the element you’re looking for, compared to the hash table where it generally only takes 1 operation.